1 Warmup
   • Buggy Code
   • Valgrind Output
   • Code Fix

2 Valgrind
   • Valgrind Usage

3 Types of Errors
   • Uninitialized Memory
   • Invalid Reads/Writes
   • Illegal Frees
   • Memory Leaks

4 Section Exercise
```c
#include <stdio.h>
#include <stdlib.h>

// Returns an array containing \([n, n+1, \ldots, m-1, m]\). If \(n > m\), then the
// array returned is \([\]\). If an error occurs, NULL is returned.
int *RangeArray(int n, int m) {
    int length = m-n+1;

    // Heap--allocate the array needed to return.
    int *array = (int*) malloc(sizeof(int)*length);

    // Initialize the elements.
    for (int i = 0; i <= length; ++i)
        array[i] = i+n;

    return array;
}

// Accepts two integers as arguments
int main(int argc, char *argv[]) {
    if (argc != 3) return EXIT_FAILURE;
    int n = atoi(argv[1]), m = atoi(argv[2]); // Parse cmd--line args.
    int *nums = RangeArray(n, m);

    // Print the resulting array.
    for (int i = 0; i <= (m-n+1); ++i)
        printf("%d ", nums[i]);
    puts("\n");

    return EXIT_SUCCESS;
}
```
Valgrind Output

```
==22891== Command: ./warmup 1 10
==22891== Invalid write of size 4
==22891== at 0x400616: RangeArray (warmup.c:14)
==22891== by 0x400683: main (warmup.c:22)
==22891== Address 0x51d2068 is 0 bytes after a block of size 40 alloc'd
==22891== at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==22891== by 0x4005EC: RangeArray (warmup.c:10)
==22891== by 0x400683: main (warmup.c:22)
==22891==
==22891== Invalid read of size 4
==22891== at 0x4006A5: main (warmup.c:26)
==22891== Address 0x51d2068 is 0 bytes after a block of size 40 alloc'd
==22891== at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==22891== by 0x4005EC: RangeArray (warmup.c:10)
==22891== by 0x400683: main (warmup.c:22)
==22891==
==22891== HEAP SUMMARY:
==22891== in use at exit: 40 bytes in 1 blocks
==22891== total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==22891==
==22891== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==22891== at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==22891== by 0x4005EC: RangeArray (warmup.c:10)
==22891== by 0x400683: main (warmup.c:22)
==22891==
==22891== LEAK SUMMARY:
==22891== definitely lost: 40 bytes in 1 blocks
==22891== indirectly lost: 0 bytes in 0 blocks
==22891== possibly lost: 0 bytes in 0 blocks
==22891== still reachable: 0 bytes in 0 blocks
==22891== suppressed: 0 bytes in 0 blocks
==22891==
==22891== For counts of detected and suppressed errors, rerun with: -v
==22891== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 3 from 3)
```
#include <stdio.h>
#include <stdlib.h>

// Returns an array of [n, n+1, ..., m−1, m]
// If n > m, then the array returned is [].
// If an error occurs, NULL is returned.
int *RangeArray(int n, int m) {
    int length;
    int *array;

    // XXX We must check this explicitly.
    if (n > m)
        return (int*)malloc(0);

    // Heap—allocate the array needed to return.
    length = m−n+1;
    array = (int*)malloc(sizeof(int)*length);

    // XXX We need to check if malloc returned successfully.
    if (array == NULL)
        return NULL;

    // Initialize the elements.
    // XXX We had an off—by—one error here.
    for (int i = 0; i < length; ++i)
        array[i] = i+n;

    return array;
}
```c
int main(int argc, char *argv[]) {
    if (argc != 3) return EXIT_FAILURE;
    int n = atoi(argv[1]), m = atoi(argv[2]);
    int *nums = RangeArray(n, m);

    // XXX Terminate program with failure if RangeArray cannot allocate and initialize the array.
    if (nums == NULL)
        return EXIT_FAILURE;

    // Print the resulting array.
    // XXX We had another off-by-one error here.
    for (int i = 0; i < (m-n+1); ++i)
        printf("%d ", nums[i]);
    puts(");

    // XXX Free storage before terminating.
    free(nums);
    return EXIT_SUCCESS;
}
```
Why Valgrind?

- Use of uninitialized memory
- Reading/writing memory after it has been freed
- Reading/writing off the end of malloc’d blocks
- Reading/writing inappropriate areas on the stack
- Memory leaks – where pointers to malloc’d blocks are lost forever
- Mismatched use of malloc/new/new[] vs free/delete/delete[]

These errors usually lead to crashes.
Basic Valgrind Usage

Command

`valgrind ./a.out`

Example Output

```
==26428== Memcheck, a memory error detector
==26428== Copyright (C) 2002-2011, and GNU GPL’d, by Julian Seward et al.
==26428== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==26428== Command: ./a.out
==26428==
.................... LOTS OF ERRORS ....................
==26428==
==26428== HEAP SUMMARY:
==26428== in use at exit: 528 bytes in 22 blocks
==26428== total heap usage: 22 allocs, 0 frees, 528 bytes allocated
==26428==
==26428== LEAK SUMMARY:
==26428== definitely lost: 408 bytes in 11 blocks
==26428== indirectly lost: 120 bytes in 11 blocks
==26428== possibly lost: 0 bytes in 0 blocks
==26428== still reachable: 0 bytes in 0 blocks
==26428== suppressed: 0 bytes in 0 blocks
==26428== Rerun with --leak-check=full to see details of leaked memory
==26428==
==26428== For counts of detected and suppressed errors, rerun with: -v
==26428== Use --track-origins=yes to see where uninitialised values come from
==26428== ERROR SUMMARY: 65 errors from 16 contexts (suppressed: 3 from 3)
```

- Note: Compile your C code with the GCC’s -g option for debugging information.
- Note: Valgrind accepts flags --leak-check=full and --show-reachable=yes to output more details.
Reading Uninitialized Memory

### Code

```c
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int *x;
    *x = 4; // XXX Using x before initialized.
    return EXIT_SUCCESS;
}
```

### Valgrind Output

```
==2205== Use of uninitialised value of size 8
==2205==     at 0x4004AB: main (error.c:4)
```
Illegal Reads/Writes

Code

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    int *x = (int *)malloc(sizeof(int));
    x += 2; // x now points to invalid memory (some random location).
    printf("%d\n", *x); // XXX Reading to an invalid location of memory.
    *x = 4; // XXX Writing to an invalid location of memory.
    free(x - 2);
    printf("%d\n", *((int *)3838338)); // XXX And even worse read.
    return EXIT_SUCCESS;
}
```

==3023== Invalid read of size 4
==3023== at 0x400592: main (error.c:6)
==3023== Address 0x51d2048 is 4 bytes after a block of size 4 alloc'd
==3023== at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3023== by 0x400584: main (error.c:4)
==3023==
==3023== Invalid write of size 4
==3023== at 0x4005A9: main (error.c:7)
==3023== Address 0x51d2048 is 4 bytes after a block of size 4 alloc'd
==3023== at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3023== by 0x400584: main (error.c:4)
==3023==
==3023== Invalid read of size 4
==3023== at 0x4005C4: main (error.c:9)
==3023== Address 0x3a9182 is not stack'd, malloc'd or (recently) free'd
Illegal Frees

**Code**

```c
#include <stdlib.h>

int main(int argc, char *argv[]) {
    free((void*) 0xdeadbeef); // XXX free some random address free'd.

    int *x = (int*) malloc(sizeof(int));
    free(x+4); // XXX free outside malloc'd block.
    free(x);

    return EXIT_SUCCESS;
}
```

**Valgrind Output**

```
==2978== Invalid free() / delete / delete[] / realloc()
==2978==    at 0x4C29A9E: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==2978==     by 0x400544: main (error.c:3)
==2978==  Address 0xdeadbeef is not stack'd, malloc'd or (recently) free'd
==2978== Invalid free() / delete / delete[] / realloc()
==2978==    at 0x4C29A9E: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==2978==     by 0x400562: main (error.c:6)
==2978==  Address 0x51d2050 is 12 bytes after a block of size 4 alloc'd
==2978==     at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==2978==     by 0x400562: main (error.c:5)
```
Memory Leaks

Code

```c
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]) {
    int *x = (int *)malloc(sizeof(int));
    *x = 4;
    printf("%d\n", *x);
    return EXIT_SUCCESS; // XXX Oh no! We didn’t free x.
}
```

Valgrind Output

```
==3093== HEAP SUMMARY:
==3093== in use at exit: 4 bytes in 1 blocks
==3093== total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==3093== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==3093== at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3093== by 0x400544: main (error.c:3)
==3093== LEAK SUMMARY:
==3093== definitely lost: 4 bytes in 1 blocks
==3093== indirectly lost: 0 bytes in 0 blocks
==3093== possibly lost: 0 bytes in 0 blocks
==3093== still reachable: 0 bytes in 0 blocks
==3093== suppressed: 0 bytes in 0 blocks
```
Find a partner to work with!

Look at the expandable vector code in imsobuggy.c.

First, try to find all the bugs by inspection.

Then try to use Valgrind on the same code.

The code is located at: http://www.cs.washington.edu/education/courses/cse333/13sp/sections/imsobuggy.c